NPS-CS-08-012

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

Use of Trusted Software Modules for Emergency-Integrity Display

by

Timothy E. Levin, Thuy D. Nguyen, Paul C. Clark, Cynthia E. Irvine, David J. Shifflett, Timothy M. Vidas

June 2008

Prepared for NSF and DARPA

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **01 JUN 2008**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**

4. TITLE AND SUBTITLE **Use of Trusted Software Modules for Emergency-Integrity Display**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School Center for Information Systems Security Studies and Research (NPS CISR) 1411 Cunningham Rd., Monterey, CA 93943**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**SAR**	**22**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

This report was prepared for and funded by National Science Foundation and the Defense Advanced Research Projects Agency.

This report was prepared by:

Timothy E. Levin
Research Associate Professor

Reviewed by:

Peter J. Denning
Department of Computer Science

Released by:

Dan C. Boger
Interim Vice President and
Dean of Research

This page left intentionally blank

REPORT DOCUMENTATION PAGE	Form approved OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** 21 November 2007	**3. REPORT TYPE AND DATES COVERED** Research; 7/1/07 – 7/1/08

4. TITLE AND SUBTITLE Use of Trusted Software Modules for Emergency-Integrity Display	**5. FUNDING** Grant number: CNS-0430566 and CNS-0430598
6. AUTHOR(S) Timothy E. Levin, Thuy D. Nguyen, Paul C. Clark, Cynthia E. Irvine, P. David J. Shifflett, Timothy M. Vidas	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Center for Information Systems Security Studies and Research (NPS CISR) 1411 Cunningham Rd., Monterey, CA 93943	**8. PERFORMING ORGANIZATION REPORT NUMBER** NPS-CS-08-012
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Science Foundation, 4201 Wilson Blvd. 1175 N. ArlingtonVA22230 DARPA, 3701 Fairfax Drive, Arlington, VA 22203	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER** Not applicable

11. SUPPLEMENTARY NOTES

The views expressed in this report are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT (Maximum 200 words.)

This report provides summary of the interface, mechanisms and semantics for high integrity display of information in a secure computer system, based on the use of a high assurance separation kernel and trusted software modules in both the application domain and the trusted software domain.

14. SUBJECT TERMS Operating systems: Separation Kernel; secure display; trusted software module; security; security architecture	**15. NUMBER OF PAGES** 22
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NAVAL POSTGRADUATE SCHOOL
CENTER FOR INFORMATION SYSTEMS SECURITY STUDIES AND RESEARCH

CENTER OF ACADEMIC EXCELLENCE

SecureCore

Trustworthy Commodity Computation and Communication

Technical Report: NPS-CS-08-012

Use of Trusted Software Modules for High Integrity Data Display

Timothy E. Levin, Thuy D. Nguyen, Paul C. Clark, Cynthia E. Irvine, David J. Shifflett and Timothy M. Vidas

June, 2008

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0430566 with support from DARPA ATO. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or of DARPA ATO.

Author Affiliations

Timothy E. Levin, Thuy D. Nguyen, Paul C. Clark, Cynthia E. Irvine, David J. Shifflett and Timothy M. Vidas:

Naval Postgraduate School
Center for Information Systems Security Studies and Research
Computer Science Department
Naval Postgraduate School
Monterey, California 93943

Abstract

This report provides summary of the interface, mechanisms and semantics for secure display of information in a secure computer system based on the use of a high assurance separation kernel and trusted software modules in both the application domain and the trusted software domain.

This page is intentionally blank.

Use of Trusted Application Modules for High Assurance Data Display[1]

Timothy E. Levin, Thuy D. Nguyen, Paul C. Clark, Cynthia E. Irvine,
David J. Shifflett and Timothy M. Vidas

During emergencies, there may be a need for applications to have direct access to the TCB interface in order to provide assurance that their output is securely displayed to the user. In the SecureCore architecture, an SP-based trusted software module (TSM) provides a context for high integrity code execution in the application domain, but requires underlying support for trusted display. We describe a Trusted Application Display conduit between an application-level TSM and the system display device that provides high assurance that when the submitted data is displayed, it is done so correctly, and that preserves the confidentiality and certain integrity properties of the data against attacks from the client OS and its applications.

As the data traverses through different software components, its integrity and confidentiality are cryptographically protected by SP hardware mechanisms and the TML. However, availability and context of the data depends on the TSM's processing environment including the ability of the TSM to execute without attacks on its code, data buffers, or communications to the TML. Our analysis shows that the Trusted Application Display conduit can protect the confidentiality of the data, as well as protect it from *direct* modification. With freshness validation included in the message protocol, and TSM interfaces appropriately restricted from affecting message display, the correct *context* of the data submitted to the TCB can be ensured. However the conduit cannot ensure availability in the face of a misbehaving client OS or application. Before describing the design of this capability, this report briefly reviews the SecureCore architecture, including the management of devices and communication channels.

SecureCore Overview

The trusted computing base of the SecureCore *transient trust architecture*[5] is provided by the SP processor[6][2], the Trusted Management Layer (TML), a Trusted Executive (TE) that provides execution services for high integrity applications, and the Trusted Path Application (TPA); see Figure 1.

---

[1] This document describes an on-going research effort of the SecureCore project [4], and assumes readers are somewhat familiar with the SP hardware [6] and SecureCore software architecture [5][1].

The SP processor provides tamper detection of trusted software modules (TSMs) and TSM-designated in-cache data. Certain SP registers are available only to TSMs, which can use them to store cryptographic keys. The DRK register is available only to TSMs, and only indirectly through the sp_derive instruction, which hashes data with the DRK.

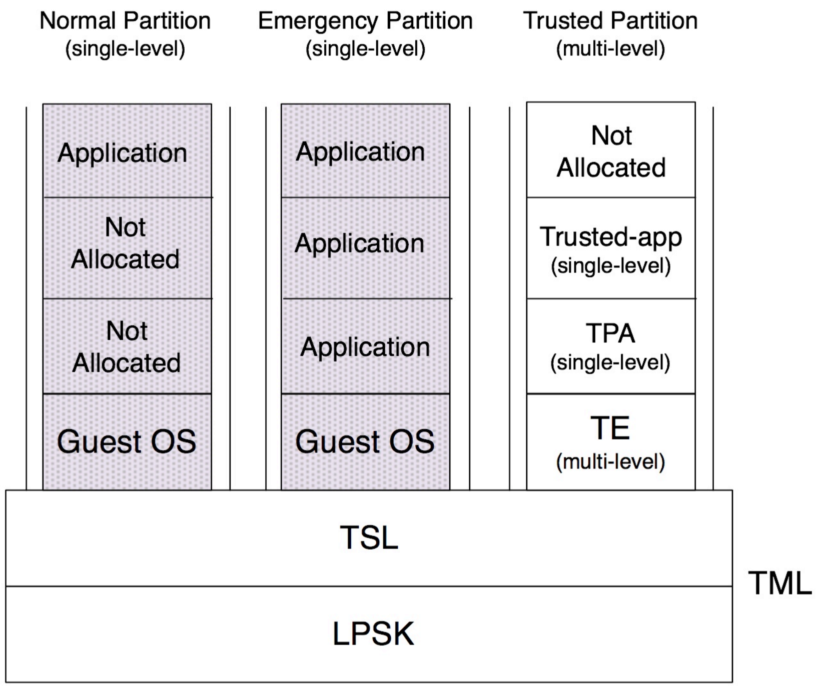


Figure 1. Transient Trust Architecture

The TML is composed of the Least Privilege Separation Kernel (LPSK) [8][1][12] and a trusted security services layer (TSL). The LPSK controls all system resources, and exports a subset of them for the use of OSes and other applications.

The TSL virtualizes certain LPSK resources and applies human readable labels to partitions. It is also responsible for emergency state management, in which it establishes trusted communication channels with the central authority, detects and verifies emergency signals from the central authority, and ensures that the user can access the emergency partition only during an emergency.

The TML provides a *partitioned* processing environment [13], much like a hypervisor, in which a separate client OS in each partition can manage its own applications without interference from other partitions. In this environment there are three types of partitions: *trusted, emergency,* and *normal*.

The trusted partition hosts a minimal Trusted Executive (TE) that provides execution services for high integrity applications, such as the Trusted Path Application. The secure attention key (SAK) keystroke sequence allows users to invoke the Trusted Path Application (TPA). The LPSK services the SAK interrupt and passes control to the TSL, which invokes the Trusted Path Application in the trusted partition.

The emergency partition provides an isolated environment for processing of sensitive information during emergencies; and normal partitions host commercial client OSes and commercial applications for day-to day use.

The TML manages a small, reserved area of the screen, for display of system messages. The messages must be short and non-graphical to accommodate the limited screen area and the simplicity of mechanisms required of high assurance TCBs. The new Trusted Application Display conduit, described below, uses this reserved area.

The TML exports both synchronous and asynchronous devices. Each exported device has two labels associated with it: a read-class and a write class.[9] A device where both labels are equal is a *single-level device*; a device for which the two labels are different is a *multilevel device*.

Management of Devices and Communication Channels

Device management is characterized in three ways. First, certain devices may be virtualized by the TML for "concurrent"[2] use by two or more partitions, which can have different security levels. The TML presents the client OSes with a virtual hardware device, and prevents undesirable interference between partitions, while the client operating systems (including the TE) may simultaneously share the services of the physical device.

Depending on the functional requirements, a concurrent-use device can be multi-level, or single-level (e.g., all partitions can read from a system low device, but if both reading and writing are required by partitions at different sensitivity levels, a multilevel device is required).

Second, a device may be assigned solely to a particular partition, in which case a single-level device at the level of the partition is usually used. For example, memory-mapped devices may be used so that they interact with processes through dedicated memory regions. Here device management is vectored by the LPSK to the client OS for its exclusive use.

Finally, devices can be allocated to one partition at a time, serially, in which case a multilevel device is used that features a "current level" (and strict object remnant cleanup). Administrators can change the current level within the device's label range, such that the device effectively has a "single level at a time." Using this device abstraction, the TML supports *I/O focus management* by allowing a given partition temporary exclusive access to the keyboard, mouse, and screen (except for the reserved screen area).

The user can change focus via the TPA interface, which allocates the keyboard, mouse and screen to a different partition. Partition multitasking is independent from focus changes, so while keyboard and mouse input is vectored to the selected partition, all partitions can continue to update their screen buffers. When the user changes focus, the TML first updates the screen with the new partition's screen buffer before passing control to the programs in the partition.

Both "serial" and "concurrent" forms of multilevel devices that support single-level partitions appear to (the processes in) each partition as single level devices, since: (1) subjects in a given partition can only read data from the device that is labeled at or below the level of that partition

[2] Either via multitasking or through multi-core or multiple processor technology

(assuming the LPSK is configured to enforce typical MLS information flow rules); and (2) data written to the device by those subjects is labeled by the TSL at the level of their partition. Note that another form of multilevel device supports concurrent use by multilevel partitions, in which case applications that write data to the device also must provide a label for each datum, and the applications are trusted to apply the correct labels.

The system also supports a configuration with tiled windowing, so that output from partitions with sensitivity levels dominated by the user's current session level can be simultaneously displayed to respective partition windows. For partitions with a level higher than the current session level, the TSL will maintain their screen output until the user establishes a session at a high enough level. Thus, instead of having to shut down activity at a particular session level to make the device available at a different level, e.g. [10][11], all partitions can continue to execute according to the predefined (multitasking) schedule.

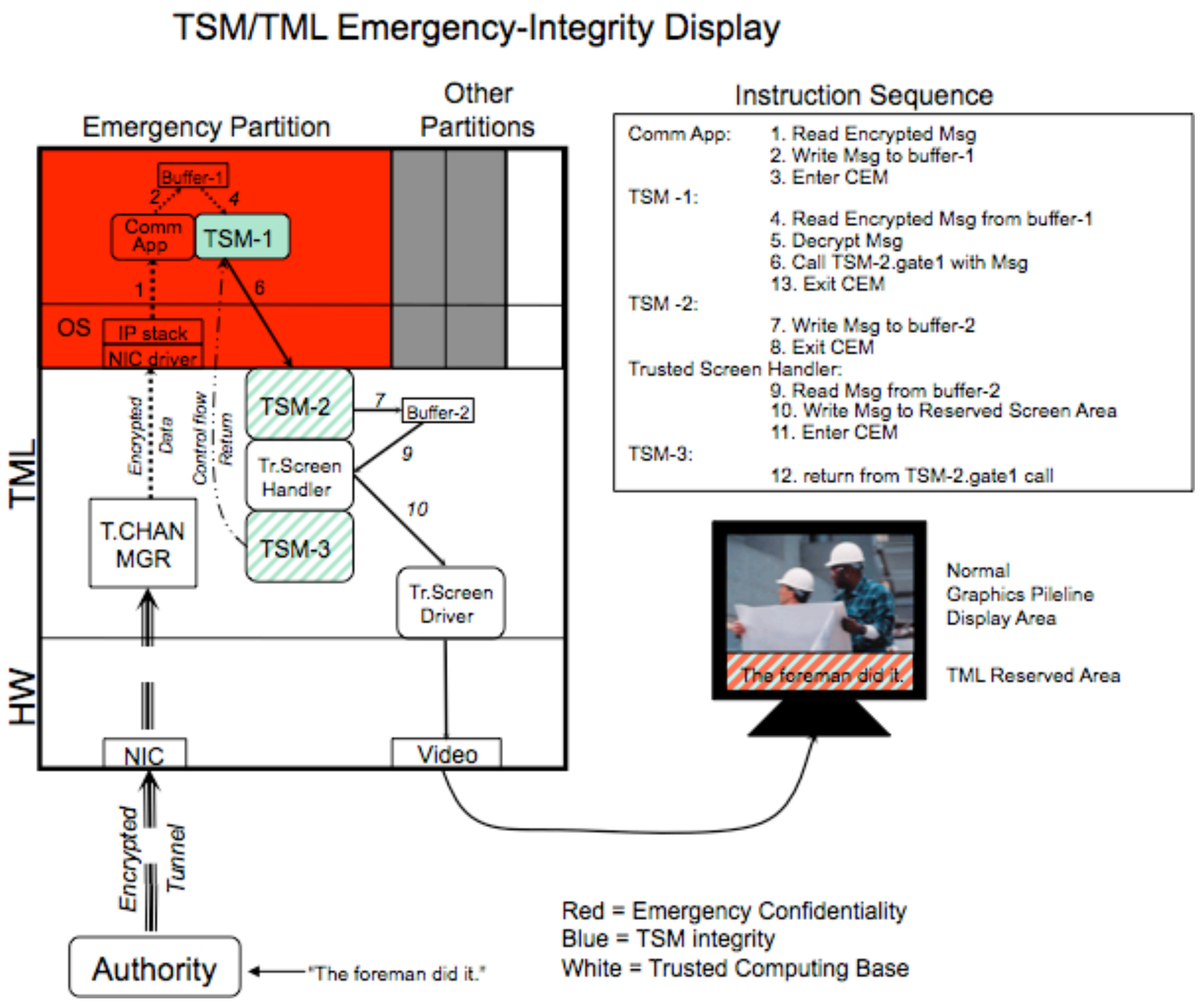
Partitions may host network-capable client operating systems. The TML multiplexes communications among its partitions and to external nodes while ensuring enforcement of the system's information flow policy. It associates implicit or explicit labels with information based upon the properties of the communicating entities. It also provides support for the establishment of secure communication channels, e.g. VPNs, with external nodes. For example, since network communication protocols are bidirectional, in a partition-to-partition communication channel, both partitions must be the same sensitivity level or, in the case of multilevel partitions, have transmission and reception ranges commensurate with the intended MLS policy.

High Integrity Display of Data

The Trusted Application Display mechanism (conduit) is exported to applications in normal, emergency and trusted partitions, and provides a simple way for information to be securely displayed without the need for the user to invoke a trusted display program.

Figure 2 illustrates how the conduit can be configured for access by an application TSM in an Emergency partition that interacts with a companion TSM in the TML. The TSM communicates with a remote central authority (the Authority) through a channel that is encrypted with the SP hardware-based Device Root Key (DRK).[6][2] The application TSM can invoke the available services of the client OS or the TML – in particular it can access the TML's trusted display conduit.

Information is first received through a trusted channel managed by the TML (see Figure 2), and is then passed via the client OS to the Comm App. The Comm App stores the information in *buffer 1*, and enters CEM mode as part of invoking the TSM (steps 1 through 3). The TSM reads and decrypts the data using SP-protected keys (4 and 5), and then stores the clear text in SP-protected memory using SP Secure_Store instructions, to prevent other applications from viewing or modifying it. Cryptographic validation of data integrity is performed (the validation protocol is outside the scope of this report) and if the validation fails, a resend of the message is requested, or the TSM exits with an appropriate error message. The application TSM then invokes a TML TSM module, through a TML call gate (6). The TML TSM reads the clear text using SP Secure_Load and then uses regular store instructions to write the data to a TML buffer (*buffer 2*) (7). The TML TSM then exits CEM mode (8) and invokes the TML's Trusted Screen Handler, which sends the data to the Trusted Screen Driver for display in the restricted display region,



Comm App:	1. Read Encrypted Msg
	2. Write Msg to buffer-1
	3. Enter CEM
TSM -1:	
	4. Read Encrypted Msg from buffer-1
	5. Decrypt Msg
	6. Call TSM-2.gate1 with Msg
	13. Exit CEM
TSM -2:	
	7. Write Msg to buffer-2
	8. Exit CEM
Trusted Screen Handler:	
	9. Read Msg from buffer-2
	10. Write Msg to Reserved Screen Area
	11. Enter CEM
TSM-3:	
	12. return from TSM-2.gate1 call

Figure 2. Trusted Application Display

labeling the data as appropriate (9,10). Finally, the TML TSM code re-enters CEM mode (11) and returns execution control to the application-TSM (12), with a return value or similar mechanism indicating the success or failure of the display operation.

Only certain partitions will be configured to have access to the Trusted Application Display interface. Least privilege within a partition is provided in different ways: LPSK subject-resource restrictions are used in partitions that utilize kernel-based processes; in the Emergency partition, only TSMs can utilize the interface, as it vectors into another TSM.

The reserved area of the screen is divided into a system area and an application area, so that messages from the TML and from applications can be displayed simultaneously. Where multiple partitions have the capability to use the application portion of the reserved screen, they must synchronize their use to avoid user confusion. For example, if an application from a high sensitivity partition is accessing the reserved area, the TML cannot block a low sensitivity application from writing to the reserved area, as a covert channel would result.

Security Analysis

High assurance computer systems provide a trusted path for users to securely interact with the trusted computing base (TCB), including a keyboard for user input and a display device for the TCB to output information to the user. The TCB may also provide interfaces for programs to write to a display device under its control such that there is high assurance that data written to the interface is displayed to the user without modification or leakage to other programs. Such assurance can be established through the program's direct access to a securely designed TCB interface, along with the ability of the TCB to protect itself. The Trusted Application Display conduit provides such a direct interface. However, this type of arrangement does not necessarily address the security of data – confidentiality, integrity, and availability – before it gets to the TCB interface.

Components that can modify an object can lower its integrity *value*[3] if they corrupt it. Given an execution path of components through which an object is passed, the object's integrity might be lowered by any component in the path that might modify, create or delete it, <u>or control such operations</u>. Even if a component does not modify the data directly, it may request that the modification be done by another component. Since the requesting component might request the wrong modification (e.g., in terms of related processing parameters), we consider that it might lower the object's integrity. Similarly, the availability of data is part of its integrity value in some contexts (e.g., a jittery picture that results from uneven signal availability has low integrity – or fidelity to the original). Even a high assurance computer system is powerless to protect the integrity of data that can be modified by untrusted user applications.

A component may be trusted – and deemed *trustworthy* – to only modify objects as, if, and when it is directed to do so. So, we say that the integrity of a modifiable object can be no higher than the greatest lower bound of the integrity, or trustworthiness, of the components on an execution path through which it is passed.[3] But, how trustworthy is a component? A component can be *evaluated* to determine how well it conforms to its specification and is free of hidden functions (here, "evaluated" means that the software is certified by a third party to have a level of assurance commensurate with the integrity of the data it is assigned to handle). However, components may depend on other components to provide functions, services, and environmental assurances, so additionally, a component can be trustworthy only to the extent that the components it depends on are trustworthy.[7] A key to this analysis is that the application TSM does not depend on the untrusted client OS functionally, as it does not "call" the client OS.

In a separation kernel (or "hypervisor") environment in which the separation kernel comprises the TCB, the execution path can involve a remote computing environment and the internet (if the data originates remotely), the TCB, the client OS, the program that writes directly to the TCB interface, as well as other applications (e.g., the Comm App in Figure 2, or a user interface app if the data originates with the local user). Additionally, applications that are not in the (intended) data path can potentially affect the security of the data.

The security of remote processing and network transmission are outside of the scope of this report, although technology for that security probably exists (e.g., through cryptographic

[3] As opposed to its integrity *label*

protection of IP packets). We also assume that the data, as created, reflects the creator's intent. With respect to local components:

a) We assume that the TCB is validated to be correct and self-protecting.[12]

b) In the SecureCore architecture, the SP processor protects TSM applications at the cache-line level: if they have been modified, the processor will not allow their execution. We assume that the submitting program is a TSM and is, a priori, correct (e.g., has been completely evaluated by a third party), and that if it executes, it does so securely (although, through no fault of its own, it may not execute or may execute too slowly to preserve the value of time-sensitive data).

This leaves only the client OS and its applications (including the Comm App) as components that could undermine the security of the data to be displayed (a component could be natively malicious or could have been corrupted so as to misbehave). The threats from misbehavior of these components are:

- Blocking the data. In this category, we include delays that invalidate the data.
- Changing the data – direct modification of the data
- Changing the context of the data submission to the TCB, such that the intent of the sender of the data is subverted. This includes corrupting related parameters, and replay of old messages.
- Leaking the data to a different partition – i.e., to a lower sensitivity level
- Leaking the data to other components within the same partition – to components within the same level that are not supposed to see the data according to the security policy allocated to the client OS.

Table 1 shows a summary of the protection provided by the Trusted Application Display conduit, in terms of the threats to the data from misbehavior of components before the program submits it to the TCB. Items lettered "c" through "g" are discussed below; other items are discussed under "a" and "b" above:

c) Block or Delay Data. The client OS and application components process the data before it is passed to the application TSM. These components can potentially delete the TSM's buffer, alter the buffer content so as cause the integrity check to fail, hog the CPU, insert previous messages, kill the TSM, etc.

   In the worst case, the message is never given to the application TSM because the components refuse to process the data or a message that fails an integrity check cannot be replaced.[4]

[4] A *replaceable* object can be made non-modifiable, in a sense, by signing or "sealing" it cryptographically, and then discarding and replacing any instantiation that does not conform to its seal. For example, the integrity of data in a virtual private network is ensured during its transit through untrustworthy intermediary systems: modification and insertion of packets is detected, and TCP guarantees that missing or discarded datagrams are resent. This is also related to previous research which

Table 1. Methods for Preventing Misbehavior of Components

	MISBEHAVING COMPONENT		
THREATS TO DATA	**Client OS & Application Components**	**Application TSM Program**	**TML (TCB)**
Block Data	(c) Efficient design of TSM; Assured protection not identified.	(b) A priori correct; protected by SP and TML	(a) A priori correct; self-protecting
Modify Data	(d) Data is signed by sender and validated by TSM; data is protected in CEM memory.	(b) A priori correct; protected by SP and TML	(a) A priori correct; self-protecting
Modify Data Context	(e) Restricted TSM and TML interfaces; Validate in-band data freshness information	(b) A priori correct; protected by SP and TML	(a) A priori correct; self-protecting
Leak Across Domains	(f) Sender encrypts data with key material shared with TSM; data is protected in CEM memory; also LPSK will stop information flow that violates policy.	(b) A priori correct; protected by SP and TML	(a) A priori correct; self-protecting
Leak Within Domain	(g) Sender encrypts data with key material shared with TSM; data is protected in CEM memory	(b) A priori correct; protected by SP and TML	(a) A priori correct; self-protecting

Other misbehavior can result in a delay of the message, through slowed TSM processing or by necessitating a resend of a corrupted message. For example, an attacker can invalidate the TSM code integrity seal, necessitating a restart of the TSM, or can invalidate the message CEM integrity seal while it is in CEM memory, or simply modify the buffer to invalidate the application-level integrity seal.

If the message is not time-sensitive, delays are not relevant and we can assume that its temporal and syntactic integrity are intact when received by the application TSM. Otherwise, time-sensitive messages can be invalidated if the original message or its replacement are not received and displayed by the TML within the message's time-sensitivity bounds.

Assuming a benign environment, the TSM must be able to process messages efficiently enough to meet the time-sensitive requirements one would expect in an emergency-response network. But, ultimately, if the OS or applications block or delay the data in the ways

enables a *low* confidentiality object to be sent into a *high* confidentiality untrusted domain and returned as *low* information, only if the object's cryptographic seal (including the low label) is intact.

described above, neither the TSM nor the underlying TML can prevent or mitigate such activity.

d) Modify Data. This threat can be addressed. Data is signed by sender and validated by the TSM. The keys for validation are protected by the TSM. TSM uses CEM memory to cryptographically protect the data when it is being processed.

e) Modify Data Context. We identify two forms of data context modification that can invalidate the integrity of the data: parameter corruption and message replay. Both of these data context threats can be addressed. If any of the parameters involved with invoking the application TSM or the TML TSM affect the content of the message to be displayed, and those parameters could be distorted by the client OS or by other applications, then the integrity of the displayed data could be compromised. For example, a parameter could call for the substitution of certain words, or determine when to display the message. To minimize these problems, we designed the interfaces to the application TSM and the TML TSM to have no parameters that affect the message content.

   If messages are intercepted, and resent in a different temporal context, they may have little integrity. It is possible that a trusted communication channel could guarantee the freshness of IP packets received by the TML, however, the client OS or applications could replay the packets. To minimize the threat of replay, we require that the message itself has freshness information that the application TSM or the user can validate.

f) Leak Across Domains. This threat has been addressed in our trusted display design, using SP and LPSK security features. The sender encrypts data with key material shared with TSM. The keying material is based on the DRK, which only the TSM can access. The data is protected during processing in CEM memory.

   LPSK will prevent any information flow between partitions that violates policy, e.g., write downs are prohibited.

g) Leak Within Domain. This threat has been addressed in our trusted display design, using SP security features. The sender encrypts data with key material shared with TSM. The keying material is based on the DRK, which only the TSM can access. The data is protected during processing in CEM memory.

   In the emergency partition, the client OS is configured with a single process and a single data segment, from which it creates and exports more granular resources. Therefore intra-partition flows between these OS exported resources are not constrained by the LPSK policy, as are leakages across domains.

Summary

This report presents an innovative high-level design to support secure display from an application software module. The design of the Trusted Application Display utilizes both hardware and software protection mechanisms to preserve data confidentiality and integrity. This capability affords a high level of confidence that the data seen on the display device has not been viewed by unauthorized subjects or tampered with by untrusted programs executing in the application domain. Our analysis also shows that with freshness validation included in the message protocol, and TSM interfaces appropriately restricted from affecting message display, the correct context of the data submitted to the TCB can be ensured; but the Trusted Application Display does not ensure availability in the face of misbehaving client OS or applications.

REFERENCES

[1] Paul C. Clark, Cynthia E. Irvine, Timothy E. Levin, Thuy D. Nguyen and Timothy M. Vidas, *SecureCore Software Architecture: Trusted Path Application (TPA) Requirements*, NPS Technical Report NPS-CS-07-001, Naval Postgraduate School, Monterey, CA, December 2007.

[2] Dwoskin, J. and Lee, R. "Hardware-rooted Trust for Secure Key Management and Transient Trust." *CCS'07*, October 2007, Alexandria, Virginia, USA

[3] Cynthia Irvine and Timothy Levin. "A Cautionary Note Regarding the Data Integrity Capacity of Certain Secure Systems." *Proceedings of the working conference on integrity and internal control* IFIP, Brussels, Belgium. November 2001

[4] Irvine, C., *Collaborative Research: SecureCore for Trustworthy Commodity Computing and Communications*, www.fastlane.nsf.gov, Award 0430566. 31 Mar. 2005.

[5] C. E. Irvine, T. E. Levin, P. C. Clark, and T. D. Nguyen, "A Security Architecture for Transient Trust," to appear in *Proc. Computer Security Architecture Workshop*, ACM, November 2008.

[6] R. Lee, P. Kwan, J. McGregor, J. Dowskin, and Z. Wang, "Architecture ror Protecting Critical Secrets in Microprocessors," in *Proc. 32nd International Symposium on Computer Architecture*, (Madison, Wisconsin), pp. 2–13, IEEE Computer Society, June 2005.

[7] T. E. Levin and C. E. Irvine and T. V. Benzel and G. Bhaskara and P. C. Clark and T. D. Nguyen. *Design Principles and Guidelines for Security*. Naval Postgraduate School NPS-CS-07-014 Monterey, CA November 2007

[8] T. E. Levin, C. E. Irvine, C. Weissman and T. D. Nguyen, "Analysis of Three Multilevel Security Architectures." in *Proc. Computer Security Architecture Workshop*, Fairfax, Virginia, USA November 2007.

[9] T. F. Lunt, P. G. Neumann, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley, *Secure Distributed Data Views Security Policy and Interpretation for DMBS for a Class A1 DBMS*, Tech. Rep. RADC-TR-89-313, Vol I, Rome Air Development Center, Griffiss, Air Force Base, NY, December 1989.

[10] National Computer Security Center, *XTS- 300 Final Evaluation Report*, Wang Federal Incorporated, CSC-EPL- 92/003.B, July 11, 1995.

[11] National Computer Security Center, *Gemini Trusted Network Processor Final Evaluation Report*, Report No. 34-94, June 28, 1995.

[12] National Information Assurance Partnership, *U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness*. Version 1.03 ed., 29 June 2007.

[13] J. Rushby, "Design and Verification of Secure Systems," in *Proc. of the 8th ACM Symposium on Operating Systems Principles*, pp. 12–21, 1981.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
2.	Dudley Knox Library, Code 013 Naval Postgraduate School Monterey, CA 93943	1
3.	Research Office Naval Postgraduate School Monterey, CA 93943	1
4.	Paul C. Clark Department of Computer Science Naval Postgraduate School Monterey, CA 93943	2
5.	Cynthia Irvine Department of Computer Science Naval Postgraduate School Monterey, CA 93943	2
6.	Timothy Levin Department of Computer Science Naval Postgraduate School Monterey, CA 93943	2
7.	Karl Levitt National Science Foundation 4201 Wilson Blvd. Arlington, VA 22230	1
8.	Thuy Nguyen Department of Computer Science Naval Postgraduate School Monterey, CA 93943	2
9.	David J. Shifflett Department of Computer Science Naval Postgraduate School Monterey, CA 93943	2

10. Timothy M. Vidas 2
    Department of Computer Science
    Naval Postgraduate School
    Monterey, CA 93943